

Passing arguments to a python script.

  
[Contents](#) [Previous](#) [Next](#)

Goal: Learn how to pass arguments to a python script.

Python let you acces whatever was passed to a script when calling it.
The "command line" content is stored in the the [sys.argv](#) list

The script described in this example is accessible [here](#)
python params.py

```
import sys
print sys.argv # returns: ['param.py']
```

You can now parse the list and check for values that the user pass.

A nice way to deal with parameters and values passed is to use the getopt module

This module helps parsing the list of arguments and returns an error if the user passed an invalid keyword.
This allows you to create "unix-like" script, e.g with the `-p` or `--parameter` options
"letters" or "parameters" can be followed or not by a value
You need to first define the valid letters, in this example we will say that the user can pass a variable or a model via the `-v/-m` letters or `--variable=/--model=` keywords
The "letters" are to be listed in a string:

```
letters = 'v:m:' # the : means an argument needs to be passed after the letter
```

Keywords are defined in a list (as they contain more than 1 letter...)

```
keywords = ['variable=', 'model=' ] # the = means that a value is expected after
# the keyword
```

Now we can have the getopt module verify that all the keyword/option passed are valid and have argmunets when necessary

```
import getopt
opts, extraparams = getopt.getopt(sys.argv[1:])
# starts at the second element of argv since the first one is the script name
# extraparms are extra arguments passed after all option/keywords are assigned
# opts is a list containing the pair "option"/"value"
print 'Opts:',opts
print 'Extra parameters:',extraparam
```

Now we simply have intialized or values and loop through the options passed by the user to see if we need to modify them

```

var='default'
model='default'

for o,p in opts:
    if o in ['-v','--variable']:
        var = p
    elif o in ['-m','--model']:
        model = p

print 'Variable:',var
print 'Model: ',model

```

Here are some examples when running this script (you can get it [here](#))

```

>python param.py
['param.py']
Opts: []
Extra parameters: []
Variable: default
Model: default
>python param.py -v tas
['param.py', '-v', 'tas']
Opts: [('-v', 'tas')]
Extra parameters: []
Variable: tas
Model: default
>python param.py --variable=tas
['param.py', '--variable=tas']
Opts: [('--variable', 'tas')]
Extra parameters: []
Variable: tas
Model: default
>python param.py --variable=tas -m modell
['param.py', '--variable=tas', '-m', 'modell']
Opts: [('--variable', 'tas'), ('-m', 'modell')]
Extra parameters: []
Variable: tas
Model: modell
>python param.py --variable=tas --model=modell
['param.py', '--variable=tas', '--model=modell']
Opts: [('--variable', 'tas'), ('--model', 'modell')]
Extra parameters: []
Variable: tas
Model: modell
>python param.py --variable=tas --model=modell extraparameters would be here
['param.py', '--variable=tas', '--model=modell', 'extraparameters',
'would', 'be', 'here']
Opts: [('--variable', 'tas'), ('--model', 'modell')]
Extra parameters: ['extraparameters', 'would', 'be', 'here']
Variable: tas
Model: modell

```

Download the [param.py](#) .



[Contents](#) [Previous](#) [Next](#)